Form Method Syst Des
DOI 10.1007/s10703-006-7843-9

A predictive synchronizer for periodic clock domains

Uri Frank · Tsachy Kapshitz · Ran Ginosar

Published online: 3 May 2006

Abstract An adaptive predictive clock synchronizer for systems on chip incorporating multiple clock domains is presented. The synchronizer takes advantage of the periodic nature of clocks in order to predict potential conflicts in advance, and to conditionally employ an input sampling delay to avoid such conflicts. The result is conflict-free synchronization with maximal throughput and minimal latency. The adaptive predictive synchronizer adjusts automatically to a wide range of clock frequencies, regardless of whether the transmitter is faster or slower than the receiver. The synchronizer also avoids sampling duplicate data or missing any input. A novel method is presented for formal treatment of synchronizers and metastability. Correct operation of the synchronizer is formally proven and verified.

Keywords Clock synchronization · Predictive synchronizer · Systems on chip (SoC) · Formal verification · Multiple clock domains (MCD) · Metastability

1. Introduction

Large systems on chip (SoC) typically contain multiple clock domains. Inter-domain communications require data synchronization, which must avoid metastability while typically facilitating low latency, high bandwidth, and low power safe transfer. The synchronizer must also prevent missing any data or reading the same data more than once.

Communicating clock domains can be classified according to the relative phase and frequency of their respective clocks [1–3]. *Heterochronous* or *periodic* domains operate at nominally different frequencies, *plesiochronous* domains have very similar clock frequencies, *multi-synchronous* domains have the same clock frequency but a slowly drifting relative phase, and *mesochronous* domains have exactly the same frequency.

The simplest solution for inter-domain data transfer is the two-flip-flop synchronizer [1, 4, 5]. The main problem with that synchronizer is its low throughput: typically, a complete

U. Frank · T. Kapshitz · R. Ginosar (✉)
VLSI Systems Research Center,
Technion—Israel Institute of Technology, Haifa 32000, Israel
e-mail: ran@ee.technion.ac.il

transfer incurs waiting about one to two clock cycles at each end, and the next transfer cannot start before that handshake is complete. Although it is a very robust solution, it is sometimes misused or even abused in an attempt to reduce its latency [6].

Another commonly used synchronizer is based on dual-clock FIFO [1, 7]. In certain situations, especially when a complete data packet of a pre-defined size must be transferred, this may be an optimal solution. Another advantage is that synchronization is safely contained inside the FIFO, relieving designers of the communicating domains of this delicate design task. The main drawback of FIFOs is their one-to-two cycle latency that is incurred when the FIFO is either full or empty, and that scenario is highly typical with periodic clock domains where the clock frequencies are different.

Mesochronous synchronizers are described in [1, 3, 8, 9]. A rational clocking synchronizer for a special case of periodic domains in which the two clocks are related by the ratio of two small integers is described in [10]. Another synchronizer for a limited case of periodic domains is described in [11]. A plesiochronous synchronizer is proposed in [12]. It incorporates an exclusion detection controlling a multiplexer that selects either the data or a delayed version of it. While having a low latency and a "duplicate and miss" algorithm for the plesiochronous case, it is inapplicable to periodic domains. Another predictive synchronizer for plesiochronous domains is presented in [13]. It predicts the transmit clock behavior compared to the receive clock in advance. Using this data an "unsafe" signal is produced to control data latching.

Dally and Poulton [1] suggest a predictive synchronizer for *periodic* clock domains in which two versions of the data are latched and selected according to the output of a phase comparator that compares the two clocks. The circuit is non-adaptive, requiring advanced knowledge of the two frequencies, and does not handle missed or duplicate data samples.

We investigate *Adaptive Predictive Synchronizers* for low-latency and maximum throughput bridging of periodic domains where the two frequencies are unknown in advance at design time and may also change from time to time. Such synchronizers are also suitable for other types of domain relationships. In particular, predictive synchronizers are designed for high performance minimal latency transfer of data almost every cycle (of the slower clock). The synchronizers must prevent missing any data or sampling the same data more than once.

In Section 2 we describe the problem and introduce the predictive synchronizer. The principal component of the synchronizer is presented in Section 3, and we analyze its exposure to the risk of metastability in Section 4. Data coherency is studied in Section 5. In Section 6 we discuss model-checker based formal verification of the synchronizer.

2. Synchronizer overview

Consider high bandwidth data sent from a transmitter clock domain to a receiver domain. The data lines change state simultaneously with the rising clock of the transmitter domain, and the transmitter clock is sent to the receiver together with the data, serving as a "data valid" or "ready" signal (this is also termed "source synchronous" data transfer). Metastability may happen at the receiving end if the receiver (sampling) clock rises simultaneously with any change of the received data.

The *Two-Way Adaptive Predictive Synchronizer* (Fig. 1) synchronizes bi-directional communications between two periodic clock domains ('Left' and 'Right'). The synchronizer receives the two clocks, and manages safe data transfers both ways. It produces SEND and RECV control outputs to both domains, indicating when it is safe to receive and send new data on both sides, avoiding data misses and duplicates due to mismatched clock frequencies.

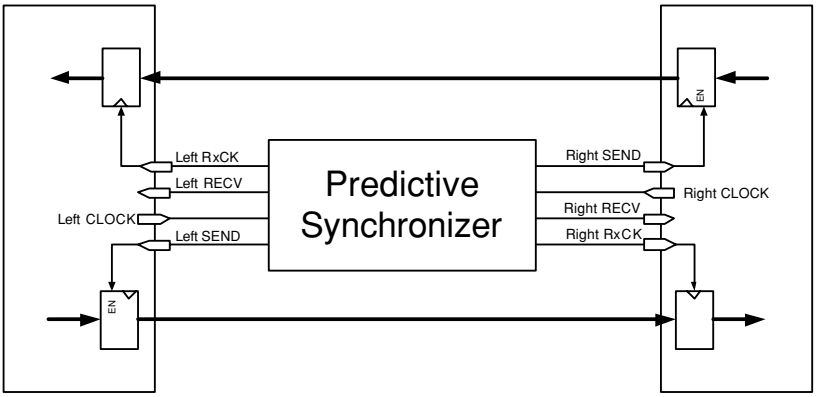


Fig. 1 A two-way predictive synchronizer with miss and duplicate protection

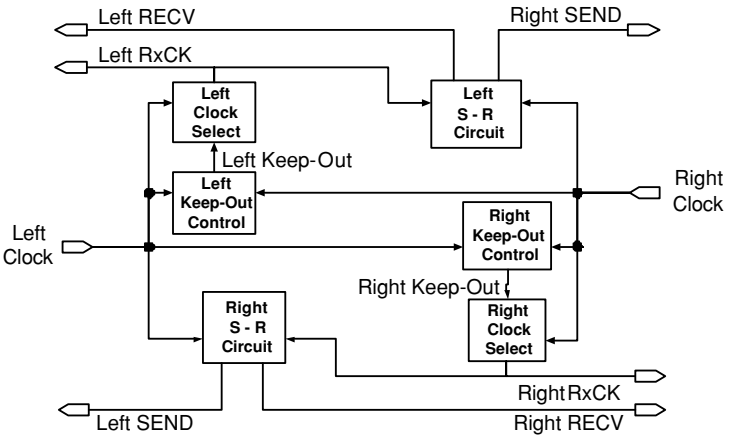


Fig. 2 Architecture of the two-way predictive synchhronizer

Figure 2 shows the internal structure of the two-way predictive synchronizer. Since it is symmetric, we consider only one half of it, the part that moves data from Left to Right. We adopt the term "Local Clock" for the receiver's clock (the Right Clock in this case) and "External Clock" for the sender's (Left) clock.

Clock conflicts can be predicted in advance, thanks to the periodic nature of the two clocks. Let T_{LOCAL} and T_{EXT} be the clock periods of the receiver (local) and transmitter (external) clocks, respectively. Let's assume that we have a conflict at time zero. The next conflict occurs when an integral number of cycles of the local clock span the same time as an integral number of cycles of the external clock, namely there should exist some N and K such that:

$$N \times T_{\mathrm{local}} = K \times T_{\mathrm{ext}}. \tag{1}$$

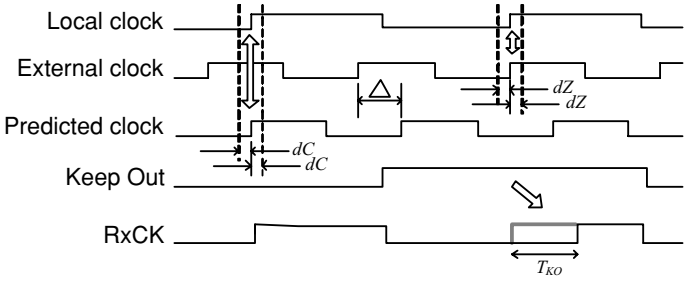


Fig. 3 A conflict is predicted one cycle in advance, delaying input sampling

The imminent conflict can be predicted as follows. Let's find the smallest Δ such that

$$T_{\text{local}} + \Delta = M \times T_{\text{ext}} \tag{2}$$

where M is an integer. Then, subtract T_{LOCAL} from each side of (1) and substitute (2):

$$(N-1) \times T_{\text{local}} = K \times T_{\text{ext}} - T_{\text{local}}$$
$$(N-1) \times T_{\text{local}} = (K-M) \times T_{\text{ext}} + \Delta$$

Thus, conflict prediction is achieved by creating a *Predicted Clock*, which is a version of the external clock delayed by Δ. The Predicted Clock and the local clock will now conflict one T_{LOCAL} cycle before the imminent conflict of the external and local clocks, allowing us to predict it. This is demonstrated in Fig. 4. A *d-conflict* is defined as the simultaneous occurrence of two clocks within time interval d of each other. Thus, when a *dC*-conflict is detected between the local and predicted clocks, we know that one (local) cycle later there is going to be a *dZ*-conflict between the local and external clocks (*dC* and *dZ* are explained below). Hence, sampling of the input (which is affected by RxCK) is delayed by a keep-out time T_{KO}, where $T_{\text{KO}} > dZ$.

Conflict prediction is performed in the *Keep-Out Control*, and is described in detail in Section 3. Detection of clock conflicts may lead to metastability, as discussed in Section 4. In addition to conflict resolution, the synchronizer must also assure data coherency. When the receiver is slower than the sender, it may occasionally miss a data item, and the sender must be stalled in such cases to prevent data misses. Conversely , when the receiver is faster, it must be stalled to avoid reading the same data item twice. This is controlled by the *S-R Circuit,* as described in Section 5.

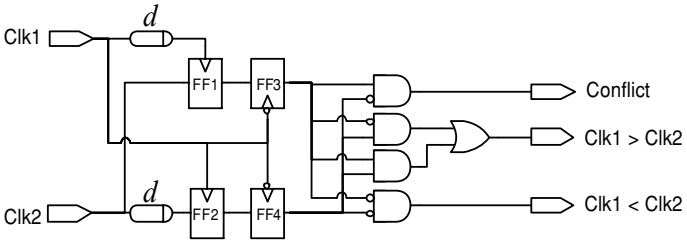


Fig. 4 *d*-conflict detector

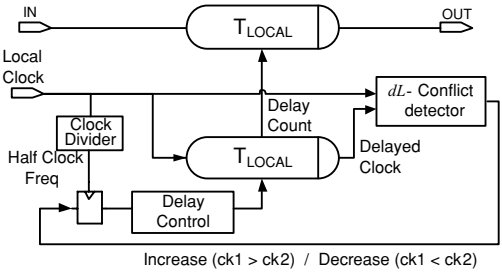


Fig. 5 The adjustable T_{LOCAL} delay line

3. Synchronizer structure and operation

In this section we describe the structure and operation of the synchronizer. Three different versions of a *conflict detector* are employed, so we start by describing it in Section 3.1. The synchronizer is capable of handling dynamic clock frequency scaling, thanks to adaptive computation of the local clock cycle (Section 3.2). The unit that computes the predicted clock is shown in Section 3.3, and the complete structure is discussed in Section 3.4.

3.1. The conflict detector

The d-conflict detector determines that two events (pre-specified rising signal transitions) either occur within d time of each other, or that one precedes the other. Figure 4 shows a conflict detector circuit. Flip-flops FF1 and FF2 effectively sample Clk2 d time after and d time before the rising edge of Clk1, respectively. Either FF may become metastable. In this circuit, one half cycle (of Clk1) is allotted for metastability resolution (as discussed in Section 4). Following the resolution interval, the outputs of FF1 and FF2 are sampled by FF3 and FF4, respectively. If Clk2 has risen during the $2d$ detection period, the top AND gate is enabled and Conflict output is generated. In other cases, the other outputs signal which input event happened first.

In Section 4 we analyze the susceptibility of the d-conflict detector to metastability. In particular, we define safety and show that the d-conflict detector safely detects conflicts. This property is used in the following sub-sections.

3.2. Computing the clock cycle time

A "T_{LOCAL}" circuit (Fig. 5) contains an adaptive delay line that is dynamically tuned to the cycle time of the local clock. It employs programmable delay lines that consist of simple digital tapped inverter chains [9, 14]. This circuit starts with a minimal delay and increases (or decreases) the delay until it is equal to a full cycle. The clock divider and flip-flop provide a loop delay (of two local clock cycles) until the lower programmable delay line has had time to adjust to a new value and the dL-conflict detector has responded to that new value. The time resolution dL of the conflict detector must be larger than the adjustment step of the delay line (typically, the delay step is about two gate delays, and dL is made at least four gate delays). Once the lower delay line has converged to T_{LOCAL}, its programming code is copied to the upper delay line. Thereafter, the circuit tracks changes in the cycle time (by means of the Increase and Decrease signals).

Theorem 1. *The T_{LOCAL} unit safely computes T_{LOCAL} with precision $\pm dL$.*

Proof: The delay line is increased from zero delay and until the *dL*-conflict detector safely indicates a conflict between the local clock and its delayed version. Since the *dL*-conflict detector can only determine that its two inputs are simultaneous within a window of *dL*, the precision of the delay is no better than $\pm dL$. □

The T_{LOCAL} unit converges to the cycle time of the local clock as follows. The minimum step by which the delay line can be increased or decreased is q. The delay line starts from zero and increases the delay by steps of size q (clearly, q must be shorter than dL) up to the complete local cycle time, taking $\lceil T_{\text{LOCAL}}/q \rceil$ steps. Each step takes two local clock cycles, and thus the total convergence time of the DLL is $\lceil T_{\text{LOCAL}}/q \rceil \times 2 \times T_{\text{LOCAL}}$.

3.3. The clock predictor

The adaptive clock predictor (Fig. 6) improves on the original design of Dally and Poulton [1]. It comprises two adjustable delay lines: The T_{LOCAL} circuit of Fig. 5, and a Δ delay line. The feedback circuit adjusts the Δ programmable delay line (cf. Eq. (2)) so that the two inputs to the *dP*-conflict detector rise within *dP* time of each other. The "predicted clock" output provides a copy of the external clock one local cycle time in advance. Once adjusted, the clock predictor tracks any delay variations to within the precision limit of $\pm(dL + dP)$, as proven below.

The loop delay in the adaptive clock predictor (the delay between successive steps of delay adjustment) must be the maximum of the two clock cycles. Since it is unknown in advance which clock is slower (either the external or the local clock), the rate reducer (Fig. 7) waits for at least one cycle of each, synchronizing to each clock in turn by means of two flip-flops. Metastability analysis of the rate reducer is similar to that of the conflict detectors, and is deferred to Section 4.

The delay introduced by the rate reducer between successive adjustments of Programmable Delay 1 (two passes around the circle in the rate reducer) is $4T_{\text{LOCAL}} + 4\text{T}_{\text{EXTERNAL}}$ (in the worst case). Programmable Delay 1 must be tuned to a total delay of $\varepsilon = \lceil K/N \rceil \times T_{\text{EXTERNAL}} - T_{\text{LOCAL}}$ (as in Section 2). Since each time the delay is increased only by q, the number of steps required to tune Programmable Delay 1 is $\frac{(\lceil K/N \rceil \times T_{\text{EXTERNAL}} - T_{\text{LOCAL}})}{q}$. As the delay between successive steps is determined by the rate reducer, the total tuning time for Programmable Delay 1 is

$$\frac{(\lceil K/N \rceil \times T_{\text{EXTERNAL}} - T_{\text{LOCAL}})}{q} \times 4(T_{\text{LOCAL}} + T_{\text{EXTERNAL}})$$

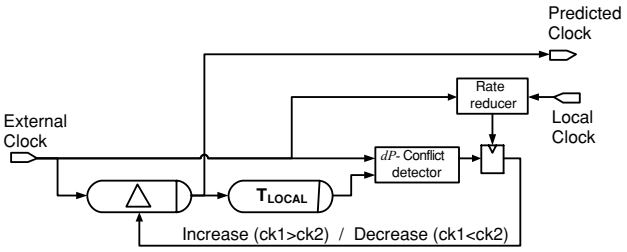
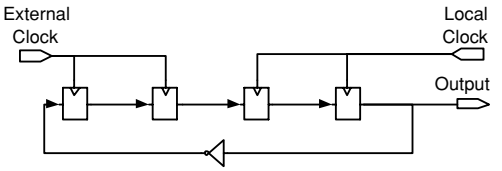


Fig. 6 Adaptive clock predictor

Fig. 7 Rate reducer



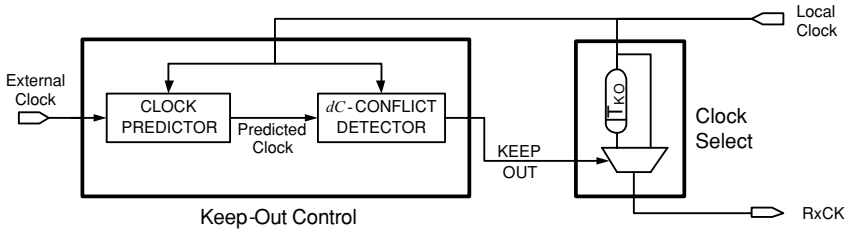


Fig. 8 Keep-Out Control and Clock Select

The total adaptation time comprises the above expression plus the convergence time of the T_{LOCAL} delay line, $(T_{\mathrm{LOCAL}}/q) \times 2 \times T_{\mathrm{LOCAL}}$.

Theorem 2. *The Clock Predictor safely generates a delayed version of the external clock that periodically precedes its original version by* T_{LOCAL} *with precision* $\pm\,(dL + dP)$.

Proof: The two delay lines in Fig. 6 are increased from zero delay and until the *dP*-conflict detector safely indicates a conflict between the external clock and its delayed version. Since the *dP*-conflict detector can only determine that its two inputs are simultaneous within a window of *dP*, the precision of the delay is no better than $\pm\, dP$. According to Theorem 1. The Predicted Clock output is generated so that there is a delay of $T_{\mathrm{LOCAL}} \pm dL$ between it and the *dP*-conflict detector. Thus, the Predicted Clock rises $T_{\mathrm{LOCAL}} \pm (dL + dP)$ prior to the original version of the external clock. □

3.4. Conflict prevention circuits

The Keep-Out Control and Clock Select are shown in Fig. 8. The *dC*-conflict detector produces the Keep-Out signal upon a *dC* conflict of the local and predicted clocks. Keep-Out is valid from one (local clock) half cycle time before the rising edge of the sampling local clock and until one half cycle later (see Fig. 3). The Clock Select circuit produces RxCK depending on Keep-Out. Thus, RxCK is either the original local clock (when there is no predicted conflict) or the T_{KO} delayed local clock (when a conflict is predicted).

The original intent of the synchronizer (Fig. 3) is to assure at least *dZ* separation between any change in the data and sampling by the receiver. That separation provides both for the setup and hold constraints for sampling the input data ($dS = \max(t_{\mathrm{SETUP}}, t_{\mathrm{HOLD}})$, accounting also for any clock jitter) and the settling time *dF* of the S-R circuit (see Section 5 below). Thus, $dZ = \max\,(dS,\ dF\ )$. To achieve that goal, conflict prediction must allow for any uncertainties in delays as well as for *dZ*, as follows.

Definitions

L1: A rising edge event of the local clock.
L2: A rising edge event of the local clock immediately succeeding *L*1 (namely, *L*2 succeeds *L*1 by one T_{LOCAL}).
P: A rising edge event of the predicted clock.
E: A rising edge event of the external clock.
t_X: Time of event *X*.

Also, select *dC* such that $dC > dL + dP + dZ$. Then,

Theorem 3. *If L2 and E occur within dZ time of each other, then L1 and P safely occur within dC of each other.*

Proof: Consider Fig. 9, which demonstrates Theorems 1 and 2, the conflict windows and the event definitions.
Given,

$$|t_{L2} - t_E| < dZ$$

Also, according to the definition,

$$t_{L2} = t_{L1} + T_{\mathrm{LOCAL}}$$

and according to Theorem 2,

$$t_E = t_P + T_{\mathrm{LOCAL}} \pm (dL + dP)$$

Substituting:

$$|t_{L1} + T_{\mathrm{LOCAL}} - (t_P + T_{\mathrm{LOCAL}} \pm (dL + dP))| < dZ$$
$$|t_{L1} - t_P| < dZ + dL + dP$$
$$|t_{L1} - t_P| < dC$$

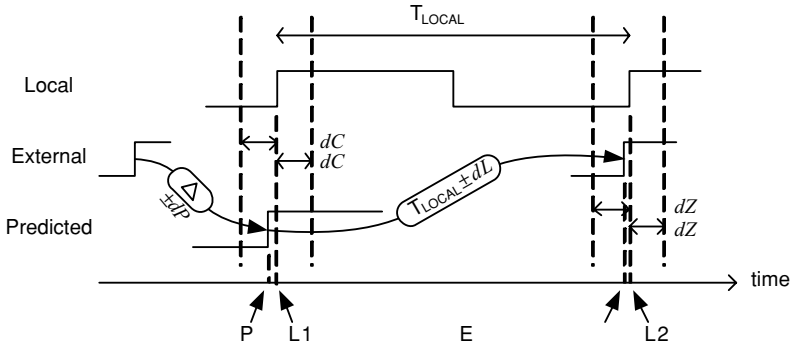


Fig. 9 Prediction timing diagram

□

Consequently, if $L1$ and P do *not* occur within dC time of each other, then $L2$ and E will be separated by at least dZ.

Now, consider the remaining case when $L1$ and P occur within dC (namely, there is a conflict and Keep-Out = 1 during $L2$), but $L2$ and E are separated by more than dZ. This could lead to the danger of conflict between RxCK (which is the delayed local clock during that cycle) and the external clock. We now show that, with a proper selection of T_{KO}, that conflict does not happen.

Definition

R: The rising edge event of RxCK.
D: Event R when Keep-Out = 1.

Theorem 4. *If L1 and P occur within dC time of each other, then D and E are safely separated by at least dZ of each other.*

Proof: Given that

$$|t_{L1} - t_P| < dC, \tag{3}$$

We need to confirm that

$$|t_D - t_E| > dZ.$$

By definition,

$$t_D = t_{L2} + T_{\mathrm{KO}}.$$

Substituting the expressions for t_{L2} and t_E from the proof of Theorem 3:

$$\begin{aligned} &|t_{L1} + T_{\mathrm{LOCAL}} + T_{\mathrm{KO}} - (t_P + T_{\mathrm{LOCAL}} \pm (dL + dP))| > dZ \\ &|t_{L1} + T_{\mathrm{KO}} - t_P| > dZ + dL + dP \\ &|t_{L1} + T_{\mathrm{KO}} - t_P| > dC \end{aligned}$$

Examining one direction of the inequality,

$$T_{\mathrm{KO}} > dC - (t_{L1} - t_P)$$

and given (3) this inequality is satisfied if

$$T_{\mathrm{KO}} > 2dC.$$

The other direction of the inequality yields irrelevant negative constraints on T_{KO}. Thus, as long as T_{KO} is selected larger than $2dC$, the theorem holds. Note also that $T_{\mathrm{KO}} < 1/2\, T_{\mathrm{LOCAL}}$ because Keep-Out is valid only until one half (local) clock cycle after $L2$ (see Fig. 3). □

In conclusion, a *dC*-conflict leads to Keep-Out=1 during *L*2, so that RxCK is the delayed local clock, and lack of *dC*-conflict leads to Keep-Out = 0 during *L*2 and RxCK is the non-delayed local clock. This can be formalized as the following theorem (easily provable):

Theorem 5. *R and E are safely separated by at least dZ time of each other.*

4. Metastability of the conflict detector

In the design presented in Section 3 above, most interactions of the two clocks (local and external) are contained inside conflict detectors. Hence, those conflict detectors may be subject to metastability. A *d*-conflict detectormay become metastable if its two input clocks rise about *d* time apart from each other. The conflict detector of Fig. 4 allows one half clock cycle for metastability resolution, and we now assess the reliability of that design.

Definition. A metastable flip-flop *resolves* when the value stored in the flip-flop is set non-deterministically to either 1 or 0, and all combinational functions of that value have been evaluated. The time to resolve a metatsable flip-flop is indeterminate and unbounded.

Definition. A circuit is said to *fail* if a combinational function of the output of a metastable flip-flop of that circuit is sampled by another flip-flop before the metastable flip-flop has resolved.

Definition. A circuit is *M-safe* if the expected time between two successive failures exceeds *M* (we use the term "safe" for short; *M* is also known as *mean time between failures*, MTBF).

Theorem 6. *A d-conflict detector is M safe.*

Proof: The asynchronous input (Clk2 in Fig. 4) goes only into the first two flip-flops. Their outputs are sampled either $T_{\mathrm{Clk1}}/2$ or $T_{\mathrm{Clk1}}/2 - d$ later. If the basic FO4 inverter delay of the implementation technology is g, the clock cycle is $T = 2k \times g$, and $d = m \times g$. Let's assume that $\tau \approx g$ and $\mathrm{W} \approx 2g$. Then (following [4], where the minimal resolution time is $T_{\mathrm{Clk1}}/2–d$):

$$M = \frac{e^{\left(\frac{T}{2} - d\right)/\tau}}{W F_C F_D} = \frac{e^{\frac{(k-m)g}{g}}}{2d \times \frac{1}{2kd} \times \frac{1}{2kd}} = 2k^2 d e^{k-m}$$

□

For instance, in SoC (according to the ITRS [15]) $2k \approx 160$. If we set $m = 10$ and choose an implementation technology of 0.18 μm where $g \approx 50$ picoseconds, then

$$M = 2 \times 80^2 \times 50ps \times e^{70} \approx 10^{16} \text{ years.}$$

M improves with modern technologies, where g is lower than 50 ps. In high-performance digital chips (such as microprocessors), the clock cycle is much shorter, e.g. $2k = 20$. To achieve reasonably high M in these cases, a different conflict detector design may be needed in which additional resolution stages are added. For instance, the conflict detector in Fig. 10

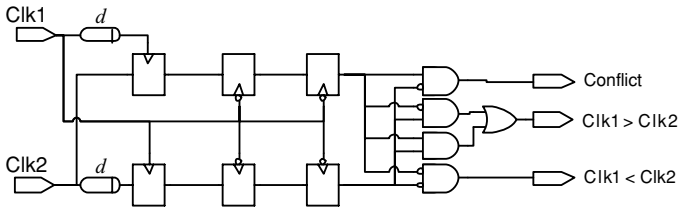


Fig. 10 Conflict detector for very fast clocks

may be employed when resolution time of 1.5 clock cycles is needed. The predictive synchronizer in such a case must be redesigned to predict the conflict *two* local clock cycles in advance, rather than one.

The only other component of the synchronizer that may become metastable is the rate reducer (Fig. 7). However, that circuit allows a complete cycle time for metastability resolution. Thus, its MTBF is e^k times that of the conflict detector.

Thus far we have shown that potentially metastable circuits in the synchronizer resolve before they are sampled (namely they do not fail) with probability that can be made as high as desirable. However, a metastable FF resolves non-deterministically to either 0 or 1. When that happens in either the *dL* or the *dP*-conflict detectors, the only effects are potential extension of convergence time or small variations in the timing of the predicted clock (where the time precision is limited to $\pm(dL + dP)$) while tracking the delays. The effect of this non-deterministic outcome in the *dC*-conflict detector is discussed in Section 5 below.

5. Avoiding misses and duplicates

When the transmitter clock is faster than the receiver's, the transmitter cannot send new data every cycle or else some data values will be missed by the receiver (Fig. 11). Conversely, when the receiver uses a faster clock, it cannot sample the input every cycle or else it will sample the same data more than once (Fig. 12).

The complete two-way predictive synchronizer provides control signals to avoid missed and duplicate data. The SEND signal guarantees (when low) that a fast sender will keep its output unchanged until it is sampled, and the RECV signal stops (when low) a fast receiver from using the same data more than once. Figure 13 specifies the algorithm that generates SEND and RECV in terms of a Signal Transition Graph (STG), where "+" and "−" mean rising and falling edge events, respectively. RECV is set upon event E (new data is available) and reset by event R (the new data has been sampled). SEND is simply the opposite of

Fig. 11 Miss condition

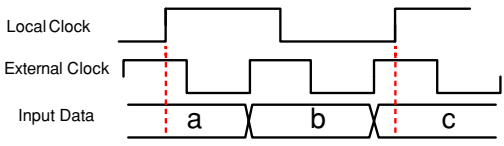
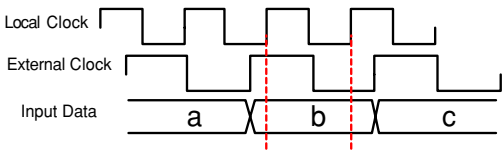


Fig. 12 Duplicate condition



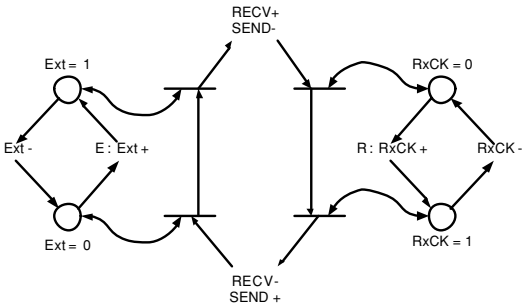


Fig. 13 SEND and RECV control STG (double arrows are probe arcs: the transition happens if the place holds a token)

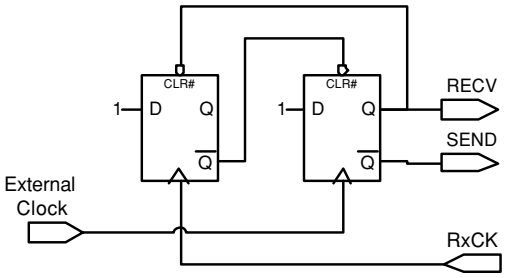


Fig. 14 Duplicate and miss control circuit

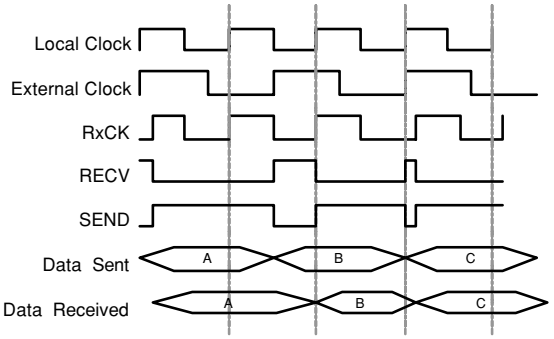


Fig. 15 Fast receiver waveforms

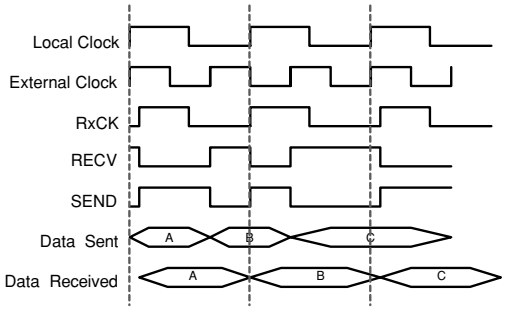


Fig. 16 Fast sender waveforms

RECV. The STG is implemented, for instance, by the circuit of Fig. 14 [16]. An example waveform of a fast receiver is given in Fig. 15 and a fast sender scenario is shown in Fig. 16.

The SR circuit requires a minimum time separation of the E and R events, in order to allow time for the following sequence of events: R → Q1+ → RECV– → Q1– → E (Q1 is a self resetting signal). This settling time was defined as *dF* in Section 3 above.

Since events E and R never conflict (Theorem 5), metastability cannot occur in the S-R circuit. Still, when the predicted and external clocks rise within *dC* time of each other, the *dC*-conflict detector may become metastable and resolve non-deterministically so that Keep-Out is either 0 or 1. This fact does not hamper correct operation, because in either case the minimum separation of *dF* between E and R is preserved, as implied by Theorems 3 and 4. However, it may result in reduced performance, especially when the two clock frequencies are the same and Keep-Out happens to oscillate due to repeating metastability events.

6. Formal verification of the predictive synchronizer

We employ model checking for formal verification of certain properties of the synchronizer [17, 18]. In general, model checking requires three components (an HDL description of the circuit, a definition of the environment, and a set of temporal logic rules on permitted event sequences) and verifies that the rules always hold. When applied to a synchronizer, we also assume that metastability issues are handled properly (e.g. as in Section 4), and a circuit that may become metastable resolves non-deterministically to either logic 1 or 0 [19].

The model checker performs checks that are separated by "ticks," where each tick is a single sequential step (and no metric time measure is associated with the ticks). In all cases below we model the two clock cycles (local and external) by means of ticks with one-tick uncertainty: T_{LOCAL} = (either m or $m+1$) ticks, and T_{EXT} = (either n or $n+1$) ticks.

The following properties of the predictive synchronizer were verified: The synchronizer correctly predicts a conflict one (local) cycle in advance (Section 6.1), it correctly generates a Keep Out signal (Section 6.2), the S-R unit correctly generates the SEND and RECV signals (Section 6.3), and data is transferred correctly and coherently between the two clock domains (Section 6.4). Note that all verifications apply to an already-calibrated synchronizer (namely, the programmable delay lines in the clock predictor are in a stable state).

6.1. Correct conflict prediction

Conflict prediction is formally defined as the property proven in Theorem 3 (if L2 occurs within *dZ* time of E, then L1 has occurred within *dC* time of P) or its equivalent form (if L1 has not occurred within *dC* time of P, then L2 should not occur within *dZ* time of E). This metric time property is converted into a rule over a sequence of events as follows. We model the independent delays *dS*, *dF*, *dL* and *dP* as a single tick each. Since $dZ = \max(dS, dF)$ and $dC > dL + dP + dZ$ (as discussed in Section 3), we model *dC* by 4 ticks. The rule can be specified as follows (in a pseudo-temporal logic notation):

$$\text{Rule CP: } \{[^*], \text{!P[3]}, \text{L \& !P}, \text{!P[3]}\} \rightarrow \{\text{!L}[^*], \text{!L \& !E}, \text{L \& !E}, \text{!E}\}$$

In words, if we observe seven consecutive ticks with no P events and with an L event in the fourth tick, then the next L event will be preceded and succeeded by ticks with no E event. This rule must hold in all possible event sequences in the circuit. Thus, if the model checker checks the implementation and finds a possible scenario where the right hand side of Rule CP fails (both L and E occur within one tick of each other) but the left hand side holds (P and L are separated by at least 4 ticks), then the clock predictor (which produces P) is incorrectly implemented.

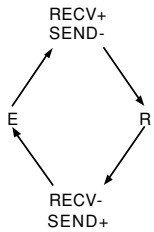


Fig. 17 Simplified STG for verification of the SR circuit

6.2. Correct keep out

Consider the following property (a derivative of Theorem 3): If L2 occurs within *dZ* time of E, then Keep-Out is asserted. Following a line of argument similar to Section 6.1, we convert that property into the temporal logic rule:

$$\text{Rule KO1: } \{[^*], \{L, E\} \textit{ or } \{L \,\&\, E\} \textit{ or } \{E, L\}\} \rightarrow \text{Keep-Out}$$

If the model checker finds a scenario where KO = 0 when the left hand side holds (L and E occur within one tick of each other) then the Keep-Out circuit is implemented incorrectly.

The other direction should also be verified: If there is no L1 and P conflict, then Keep-Out must be zero:

$$\text{Rule KO2: } \{[^*], !P[3], L \,\&\, !P, !P[3], !L[^*], L\} \rightarrow !\text{Keep-Out}$$

We can also verify that RxCK is generated correctly, namely that it never conflicts with the external clock:

$$\text{Rule RxCK: } \{[^*], \{E, R\} \textit{ or } \{E \,\&\, R\} \textit{ or } \{R, E\}\} (\text{false}).$$

6.3. Correct SEND and RECV

We now verify the operation of the SR circuit. Consider the STG of Fig. 13. The local clock is replaced by RxCK, and we know that events E and R are verified non-conflicting (as above). Hence, we can simplify the STG into Fig. 17. This STG can be converted into the following rules:

$$\text{Rule SEND: } AG(R \rightarrow !RECV \,\&\, SEND)$$

$$\text{Rule RECV: } AG(E \rightarrow RECV \,\&\, !SEND)$$

where AG stands for "always global."

The model checker fails when trying these rules on the circuit of Fig. 14. This is due to the fact that the circuit contains a self-resetting internal signal (Q1), as discussed in Section 5, and requires at least *dF* time for settling. This cannot be verified by the model checker, and is corrected by adding a "one tick delay" in the self-reset path of the verification model of the SR circuit.

6.4. Correct and coherent data transfers

We verify data transfers by considering only the data and external handshake signals (the two clocks, SEND and RECV) and ignoring the internal operation of the synchronizer. We check that the data gets transferred correctly, that no data is missed and no data is sampled more than once. This verification has initially failed, leading us to define proper conflict detection delays (*dL, dP, dC*) and other delays (*dF, dZ*), as well as their relationships, as explained and proven in Theorems 1–4 above.

Data transfer verification is applied top a single bit of the data bus, as all the other bits are chained to the same enabling logic. The following rules are employed:

(1) Each time bit 0 of the bus is sampled by the sender's register (E when SEND = 1), the next time the receiver samples that bit (L when RECV = 1) it has the same value:

$$\forall x \in \{0, 1\}\{\text{SEND, E} \,\&\, \text{bus}(0) = x\} \rightarrow \{!\text{R}[^*], !\text{R} \,\&\, \text{RECV, R} \,\&\, \text{bus}(0) = x\}$$

(2) There is at least one reception between any two successive transmissions

$$\text{AG}(\{\text{SEND, E}\} \rightarrow \text{AX}(\{\text{RECV, R}\} \text{ before! } \{\text{SEND, E}\}))$$

The "strong before" operator (with !) indicates that a transmission must be followed by a corresponding reception.

(3) There is at least one transmission between any two successive receptions

$$\text{AG}(\{\text{RECV, R}\} \rightarrow \text{AX}(\{\text{SEND, E}\} \text{ before } \{\text{RECV, R}\}))$$

7. Conclusions

A two-way adaptive predictive synchronizer for SoC with multiple clock domains has been presented. The synchronizer takes advantage of the periodic nature of clocks in order to predict potential conflicts in advance, and to conditionally employ an input sampling delay to avoid such conflicts. The result is conflict-free synchronization with almost zero latency (much less than one cycle). The adaptive predictive synchronizer adjusts automatically to a wide range of clock frequencies, regardless of whether the transmitter is faster or slower than the receiver. The synchronizer also avoids sampling duplicate data or missing any input.

Proving the correctness of synchronizers is an elusive task. Formal logic methods are challenged by the inconsistencies and non-determinism involved with synchronizers. We have created a novel formal framework for the analysis and correctness proof of complex multiple clock domain systems. Metastability is analyzed both from statistical and logic design points of view. Building on that foundation, we have been able to create correctness proof of the synchronizer. In addition, we have shown how to apply model checking for formal verification of the synchronizer design, and have discussed cases where that verification helped us improve the design.

References

1. Dally WJ, Poulton JW (1998) Digital systems engineering. Cambridge University Press
2. Messerschmitt DG (1990) Synchronization in digital system design. IEEE J. Selected Areas in Communication 8(8)
3. Ginosar R, Kol R (1998) Adaptive synchronization. Proc. ICCD
4. Dike C, Burton E (1999) Miller and noise effects in a synchronizing Flip-flop. IEEE J. Solid-State Circuits 34(6):849–855
5. Kinniment DJ, Bystrov A, Yakovlev A (2002) Synchronization circuit performance. IEEE J. Solid-State Circuits 37:202–209
6. Ginosar R (2003) Fourteen ways to fool your synchronizer. Proc. 9th IEEE Int. Symp. on Asynchronous Circuits and Systems (ASYNC03)
7. Chelcea T, Nowick SM (2001) Robust interfaces for mixed-timing systems with application to latency-insensitive protocols. Proc. ACM/IEEE Design Automation Conference
8. Chakraborty A, Greenstreet MR (2003) Efficient self-timed interfaces for crossing clock domains. Proc.9th IEEE Int. Symp. Asynchronous Circuits and Systems (ASYNC'03), pp 78–88
9. Semiat Y, Ginosar R (2003) Timing measurements of synchronization circuits. Proc. 9th IEEE Int. Symp. on Asynchronous Circuits and Systems (ASYNC'03)
10. Sarmenta LFG, Pratt GA, Ward SA (1995) Rational clocking. Proc. ICCD, pp 217–228
11. Gandhi J (2001) Apparatus for fast logic transfer of data across asynchronous clock domains. USA Patent 6,172,540
12. Dennison LR, Dennison WJ, Xanthopolous D (1995) Low-latency plesiochronous data retiming, Proc.16th Conf. Adv. Res. in VLSI, pp 304–315
13. Stewart WK, Ward S (1988) A solution to a special case of synchronization problem. IEEE Trans. Comp 37(1)
14. Moore SW, Taylor GS, Cunningham PA, Mullins RD, Robinson P (2000) Self-ccalibrating clocks for globally asynchronous locally synchronous systems. Proc. ICCD
15. International Technology Roadmap for Semiconductors (ITRS)(2001)
16. Cortadella J, Kishinevsky M, Kondratyev A, Lavagno L, Yakovlev A (1997) Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers. IEICE Trans Inform Syst E80-D(3):315–325
17. Clarke EM, Grumberg O, Peled DA (2000) Model checking. The MIT Press
18. Beer I, Ben-David S, Eisner C, Landver A (1996) RuleBase: An industry-oriented formal verification tool. Design Automation Conference pp 665–660
19. Kapschitz T, Ginosar R (2005) Formal verification of synchronizers. CCIT Tech. Rep. 536, EE Dept., Technion